ANALYSIS

46(213), October 1, 2015

ISSN 2278–5469
EISSN 2278–5450

Discovery

Bug triaging using multi-attribute bug tossing graph

Govindasamy V[1], Akila V[2], Banu Priya P[3]

1.Assistant Professor, Pondicherry Engineering College, India: Email: - vgopu@pec.edu
2.Assistant Professor, Pondicherry Engineering College, India: Email: - akila@pec.edu
3.Assistant Professor, Pondicherry Engineering College, India: Email: - banupriya.p@pec.edu

Article History
Received: 07 August 2015
Accepted: 19 September 2015
Published: 1 October 2015

Citation
Govindasamy V, Akila V, Banu Priya P. Bug triaging using multi-attribute bug tossing graph. *Discovery*, 2015, 46(213), 101-106

ABSTRACT
Bugs are non-avoidable in Open Source Software (OSS). The investigation has been done on various bug reports of OSS such as Eclipse and Mozilla. To resolve the bug, the bugs are assigned to developers. If the developer is unable to resolve it then it gets reassigned to other developer. This process is called Bug Tossing. The graph which is obtained from the above process is called Bug Tossing Graph (BTG). In the existing systems, the Bug Toss Graph is based only on the number of tosses. The resolution of bug context is a collaborative activity among developers. In this social context of the developers involved in bug resolution needs to be captured. This paper proposes a Multi-Attribute Graph using Regency, Reciprocity and Number of Tosses. A Multiple Ant Colony System (MACS) is deployed over the Multi-Attribute Bug Toss Graph. The MACS consists of two colonies which help to handle multi-objectives. Finally by implementing these two colonies, Least Crucial Graph (LCG) is obtained from the Multi-Attribute Bug Tossing Graph. In the proposed system the experimental analysis has been done by taking five year data set from www.bugzilla.org and obtained an optimal solution.

Keywords: Bug Triaging, Bug Tossing Graph, Multiple Ant Colony System.

1. INTRODUCTION

Open Source Software (OSS) is computer software in which source code is visible to everyone where anyone can enhance it or modify it. While Open Source Software (OSS) is becoming more widespread and popular, its maintenance has become an important issue. The OSS developers and users are distributed geographically and therefore management of the defect reports and source code is an important factor for the success of the project. Therefore, Open Source Software Project (OSSPs) uses at least two kinds of management systems: Defect Management System (DMS) and a Version Management System (VMS). The purpose of a DMS is to establish procedures for managing defect reports. DMSs give procedures for reporting and assigning the defect reports and modification requests. Furthermore, DMSs provide flexible possibilities for tracking and controlling defects because they store all the changes made in defect reports. VMSs establish an environment for source code management. Software Repository is a database for installing and retrieving the software. Bug tracking system, Issue Tracking system, Version Control Repositories are the examples of Software Repositories. Software repositories of large projects are typically accompanied by a bug report tracking system. Bugs are an error or flaw or fault which leads to invalid output. Bugs are non-avoidable in the software projects [1]. In an OSS, anyone can report the bug and resolve it.

Bug Repository is a database for storing and updating the information about the bugs present in the Open Source Software (OSS). Bug Triaging is the process of assigning right developers to fix a bug after identifying that the bug is not duplicate [1]. A lot of methods have been undertaken for providing automatic support for bug triaging for duplicate detection [2],[3],[4] and bug assignment[5],[6],[7],[8]. Bug fixing is crucial in producing high-quality software products. When bug(s) are filed in a bug report, assigning it to the most capable and competent developer is important in reducing the cost and time in a bug fixing process. For instance, Bugzilla (2010) is a popular bug tracker used by many large projects, such as Mozilla, Eclipse, KDE, and Gnome. These applications receive hundreds of bug reports a day; ideally, each bug gets assigned to a developer who can fix it in the least amount of time. This process of assigning bugs is known as Bug Assignment [9]. The Bug Management process allows the project team to collect, evaluate, correct, and track reported bugs and issues. When a bug is first reported, it is registered as unconfirmed and managed by Test Manager. Test Manager is responsible to periodically go through any new bugs that may have been reported and for accepting or rejecting new bug. He is also in charge to assign accepted bugs to developers and set priority and severity of the bug based, for example, on how many entities it affects and whether there are any workarounds. When a bug is fixed by a member of the development team, it is considered as resolved. The testing team uses the information provided by Reporter to verify that a bug has been fixed in a later build of the product. After successful verification, the bug is closed; however when there are still issues unresolved, the bug might be reopened for further development.

2. LITERATURE SURVEY

Senthilmani et al. [3] proposed a network analysis based approach to construct a Minimal Essential Graph (MEG) which identifies essential non-committers in a project. The non-committers here are the bug resolution catalyst. They extracted MEG as the final output from the tossing graph using tossing algorithm so that the bug triaging is significantly increased in this proposed approach. John Anvik et al. [9] presented a machine learning approach to create recommenders that help with different decisions aimed at simplify or organize the process in order to increase the efficiency in the future process. The recommenders created using this approach are minute and can be quickly configured. Pamela Bhattacharya et al. [14] proposed all-encompassing set of machine learning tools and a probabilistic graph-based model (Bug Tossing Graphs) that lead to precise predictions, and it became the foundation for the future generation of machine learning-based bug assignment. At last, they have proposed optimization techniques and achieved high prediction accuracy by reducing training and prediction time. Jin-woo Park et al. [15] modelled Content Based Recommendation (CBR) recommended only the various type of bugs that each developer had solved. This problem created difficulties in practice that some experienced developers could be overloaded with the bug and this would slow the bug fixing process. In order to overcome these difficulties two steps are taken. First, they reformulated the problem as an optimization problem of both accuracy and cost. Second, they adopted a Content Boosted Collaborative Filtering (CBCF), combining an existing CBR with a Collaborative Filtering Recommender (CFR) for the advancement of the recommendation quality.

Gaeul Jeong et al. [16] introduced a graph based model on Markov chains which captures bug tossing history. It uncover developer networks which can be used to find out the team structures and discover suitable experts for a new task and also helps to assign better developers to report the bug. Jifeng Xuan et al. [17] addressed the problem of the developer prioritization in which it aimed to rank developers contribution. It also explored two aspects, namely how to prioritize developer in bug repository and how to predict the tasks. It also analysed three problems of the developer prioritization, namely the characteristics in products, the evolution and the tolerance of noises. Liguo Chen et al. [18] modelled the tossing events which increase the bug-fix time. In order to identify the fixer quickly to report the bugs, they developed an approach to improve bug assignment with Bug Tossing Graph and bug similarities.

Experimental results evaluated based on Eclipse and Mozilla showed that this approach can reduce the bug tossing length and decrease the search failure rate.

Olga Baysal et al. [19] presented a framework for automated assignment of bug-fixing task. This approach elicits the developer to learn how to elect the bugs beforehand in order to fix it. It infers knowledge about a developer's expertise by analysing the history of bugs resolved by the developer previously. When a new bug report arrives, the system assigns it to the appropriate developer automatically considering his or her expertise, current workload and preferences. Iulian Neamtiu et al. (2010) employed several techniques to improve the triaging accuracy and to reduce tossing path lengths. Those techniques significantly reduced the bug fixing effort, especially in the context of projects with large numbers of testers and developers. Benjamin Baran et al. [20] proposed a Multi-Objective Ant Colony System for Vehicle Routing Problem with time windows. Akila et al. [21] proposed state-of-art techniques applied in bug triaging in enterprise environment and open source environment. In the related work, the Bug Toss Graph uses only the number of tosses. The social profile of the developers is not captured in the underlying data structure.

3. PROPOSED WORK

In the proposed system, a new data structure named as Least Crucial Graph (LCG). LCG is obtained from the Multi-Attributed Bug Tossing Graph. This is done by

1. Increasing the number of features for LCG
2. Multiple Ant Colony System(MACS)

In the existing system, the parameters considered are the *number of tosses*. In order to overcome the pitfalls existing in the existing system we have considered two more parameters. The parameters used in the proposed system are,

- Number of Tosses
- Recency
- Reciprocity

Number of Tosses: The number of times a bug gets tossed between tossers is denoted as Number of Tosses. First, a bug is given to a tosser, one who resolves the bug. If the tosser cannot able to resolve the bug, it will be passed to another tosser and finally at some context the bug gets resolved.

Recency: The number of days since the last communication of a contact with the tosser is called Recency. A bug is given a tosser. It is given to another tosser if he cannot resolve the bug.

Reciprocity: It isthe measurement of act of returning or responding for the bug but only with the first relationship. Then, the precision value is calculated from the Multiple Ant Colony System (MACS) using these parameters.

Multiple Ant Colony System (MACS) algorithm that has been adopted in the proposed work uses two colonies, whereas in the existing work they have used Ant Colony Optimization algorithm.

MACS is more suitable for the multi-attribute. For this reason in the proposed system MACS is used.The overall flow diagram of the proposed system is shown in the figure.

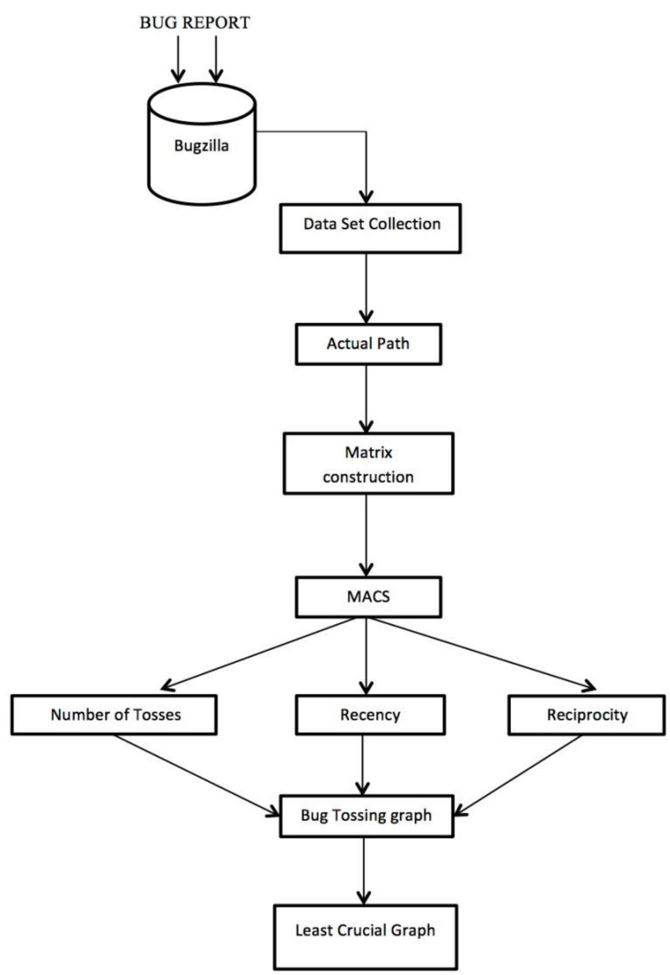

Figure 1 Flow diagram of the proposed system

4. MODULE DESCRIPTION

A. Actual Path Model Bug Toss Graph Using Markov Model:

The existing System uses only Goal Oriented Bug Toss Graphs for computing the Transition Probability Matrix. In the proposed system, both Actual Path Model Bug Toss Graphs and Goal Oriented Bug Toss Graphs are used for computing the Transition Probability Matrix which in turn is used in the process of efficient bug assignment and extracting the optimal set of resolvers. The Goal Oriented Graph Model has been already been explained in the existing System. The Actual Path Bug Toss Graph Model considered in the proposed system records the Actual Path followed in the tossing path. Consider the following Simple Tossing Path.

A -> B -> C -> D -> E

Assuming the Markov property holds, we decompose a given tossing path to several single steps

A -> B B -> C C -> D D -> E

This module computes Bug Toss Graphs (Goal Oriented Path Model and Actual Path Model) in the form of Adjacency Matrix using Markov model which is then used to compute the Transition Probability Matrix. The tossing probability is also known as the Transaction Probability. From developer D to Dj is defined by the following equation where k is the total number of developers who fixed bugs that were tossed from D:

$$\mathbf{Pr(D \rightarrow Dj) = (\#(D \rightarrow Dj)/ \sum_{i=1}^{k} \#(D \rightarrow Di))} \quad (1)$$

B. Multiple Ant Colony System (MACS):

The MACS aims to achieve the high efficiency by the two colonies namely,

1. ACS-TOS (to minimize the number of tosses)
2. ACS-TIME (to minimize the total time taken to resolve the bug).

Ant Colony Optimization is a probabilistic technique for resolving computational problems which is reduced by finding correct path via graphs. This algorithm is based on ant's behaviour which is in search of food. The ant starts their search randomly to reach the destination (food) and return to their colony by laying down the pheromone. Then the following ants also find the new path to reach the same destination. Likewise many numbers of paths will be followed by different ants. Nevertheless, the pheromone trails starts evaporating. If the path taken by the ant is longer, then the pheromone starts to evaporate faster. Therefore, the shortest path can be found out by comparing the pheromone value. This algorithm is only applicable for single objective problem which is explained in previous section. But in multi objective problems, MACS is used which handles two colonies. Each colony is helpful for optimizing two different functions. So in the proposed work, we used these two colonies for the optimization of two different parameters which is mentioned above.

Algorithm:

1. Start
2. Find A(b)=feasible solution of nth path
3. Where feasible solution = lowest number of tossers and shortest distance
4. REPEAT
5. v=active tossers
6. Activate the tossing()
   a. Minimize the number of tossers
7. Activate total time taken to resolve the bug
   a. Minimize the time taken
8. While(tossing && total time taken to resolve the bug =active)
9. Then take the improved solution A from Step5 || Step6
10. A(b)=A
11. If A(b)<v
12. Then reinitialise the tossing & total time taken to resolve the bug
13. End while
14. UNTIL the first path is reached.

5. IMPLEMENTATION AND RESULTS

The hardware requirement for the implementing the proposed system is Pentium 4 processor with 320 GB hard disc. Front end used here is Netbeans 7.2 and the Back end used is Oracle. The JDK tool is used to implement the proposed system. The bug reports of Eclipse project extracted from www.bugzilla.org during the period from 2009 to 2013 are used to validate the proposed system. The bugs with status as "RESOLVED" and "FIXED" are considered for extraction. The history of the bug is taken from the activity table. It consists of information such as who has assigned the bug, when it was assigned, report about the bug, status of the bug, next developers' name to whom the bug is assigned.

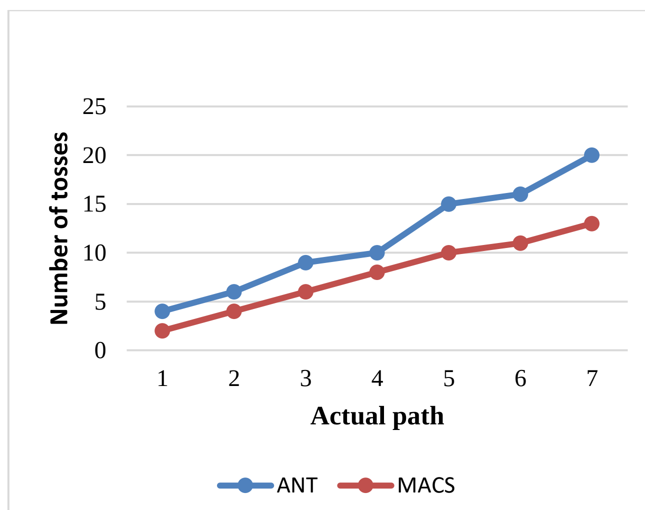

Figure 2 Least Crucial Graph

The above graph shows that the Multiple Ant Colony System (MACS) is more efficient then the Ant Routing Algorithm. The Least Crucial graph has been obtained from Multiple Ant Colony System algorithm (MACS).

6. CONCLUSIONS

We analysed history of resolved bug reports across various Open Source and different commercial projects to know and study about the roles played by different peoples in the project. Some people acted as developer and made changes in the code in order to fix the bug. In the existing work, they used only number of tossers for bug triaging. The existing system is improved by deriving a graph called Least Crucial Graph (LCG) which shows the relationship between the tossers and the path obtained. Finally we investigated and produced two more possible parameters in improving bug triaging. The first parameter finds the number of days since the last toss with the tossers (Recency) and the second is used to find the measurement of act of returning or responding only for the first relationships (Reciprocity). A graph called as Multi-Attribute Bug Tossing Graph is constructed using these parameters and the algorithm called Multiple Ant Colony System (MACS).

REFERENCE

1. D. Cubranic´, "*Automatic Bug Triage Using Text Categorization*," in In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. Citeseer, 2004.
2. G. Jeong, S. Kim, and T. Zimmermann, "*Improving Bug Triage WithBug Tossing Graphs*," in Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2009, pp. 111-120.
3. Senthil Mani, Seema Nagar, Debdoot Mukherjee, Ramasuri Narayanam, Vibha Singhal Sinha, Amit A.Nnavati, "Bug Resolution Cataysts: Identifying Essential Non-Committers from Bug Repositories"in Proceedings of the 10th working conference on mining software repositories.ACM,2015, pp. 192-202.
4. T. Zimmermann, N. Nagappan, P. Guo, and B. Murphy, "*Characterizing And Predicting Which Bugs Get Reopened*," in Software Engineering (ICSE), 2012 34th International Conference on. IEEE, 2012, pp. 1074–1083
5. P. Bhattacharya and I. Neamtiu, "*Fine-Grained Incremental Learning And Multi-Feature Tossing Graphs To Improve Bug Triaging*," in Software Maintenance (ICSM), 2010 IEEE International Conference on. IEEE, 2010, pp. 1–10.
6. L. Hiew, "*Assisted Detection Of Duplicate Bug Reports*," Ph.D. dissertation, The University Of British Columbia, 2006.
7. X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "*An Approach To Detecting Duplicate Bug Reports Using Natural Language And Execution Information*" ,in Proceedings of the 30th international conference on Software engineering. ACM, 2008, pp. 461–470.
8. P. Runeson, M. Alexandersson, and O. Nyholm, "*Detection Of Duplicate Defect Reports Using Natural Language Processing*," in Software Engineering, 2007. ICSE 2007. 29th International Conference on. IEEE, 2007, pp. 499–510.
9. J. Anvik, L. Hiew, and G. Murphy, "*Who Should Fix This Bug?*" in Proceedings of the 28th International Conference on Software engineering. ACM, 2006, pp. 361–370.
10. T. Zimmermann, N. Nagappan, P. Guo, and B. Murphy, "*Characterizing And Predicting Which Bugs Get Reopened*," in Software Engineering (ICSE), 2012 34th International Conference on. IEEE, 2012, pp. 1074–1083.
11. P. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "*Characterizing And Predicting Which Bugs Get Fixed: An Empirical Study Of Microsoft Windows*," in Software Engineering, 2010 ACM/IEEE 32nd InternationalConference on, vol. 1. IEEE, 2010, pp. 495–504.
12. Pamela Bhattacharya, Iulian Neamtiu, Christian R. Shelton,"*Automated, Highly-accurate, Bug Assignment Using Machine Learning And Tossing Graphs*",The Journal of Systems and Software 85 (2012) 2275– 2292.
13. Pamela Bhattacharya and Iulian Neamtiu,"*Fine-Grained Incremental Learning And Multi-Feature Tossing Graphs To Improve Bug Triaging*", in ICSM'10 IEEE CS, 2010.
14. John Anvik, Gail c. Murphy, *"Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions*'', Journal of ACM Transactions on Software Engineering and Methodology (TOSEM) Vol 20 Issue 3, August 2011.
15. Jin-woo Park, Mu-Woong Lee, Jinhan Kim, Seung-won Hwang, *"Costriage: A Cost-Aware Triage Algorithm For Bug Reporting Systems"*,25th AAAI Conference On Artificial Intelligence,2010.
16. Gaeul Jeong, Sunghun Kim and Thomas Zimmermann, "*Improving Bug Triage with Bug Tossing Graphs*", In Joint 12th European Software Engineering Conference (ESEC) and 17th ASM SIGSOFT symposium foundations of software engineering, pp.111-120, Aug 2009.
17. Jifeng Xuan, He Jiang, Zhilei Ren, Weiqin Zou, "*Developer Prioritization In Bug Repositories*", 34th International Conference on Software Engineering (ICSE), 2012.
18. Liguo Chen, Xiaobo Wang and Chao Liu, "*Improving Bug Assignment With Bug TossingGraphs And Bug Similarities*", International Conference of Biomedical Engineering and Computer Science (ICBECS), pp.421-425, Apr 2010.
19. Olga Baysal, Michael W. Godfrey, Robin Cohen, "*A Bug You Like: A Framework For Automated Assignment Of Bugs*", in IEEE 17th International Conference on Program Comprehension, 2009.

20. Benjamín Barán and Matilde Schaerer, "*A Multi Objective Ant Colony System For Vehicle Routing Problem With Time Windows*", in Proceedings of the 21st IASTED International Conference Applied Informatics February 10-13,2003.
21. V.Akila, G.Zayaraz, V.Govindasamy, "Bug Triage In Open Source Systems: A Review", in International Journal Of Collaborative Enterprise , vol 4,No. 4, pp 299-319.